FPGASolve

SD0917

Advisor:
Cristinel Ababei

Matthew Nitschke

Richard Schultz

Swati Gupta

FPGASolve

The FPGASolve project is designed to speed up the power flow solution using hardware acceleration, and to configure a radial distribution system for optimal losses.

Why is it needed?

- Strictly software power flow solvers are relatively slow, especially when there are many buses in the system
- This problem is compounded when attempting to obtain the optimal network configuration, because the power flow must be run many times.
- Hardware accelerator can speed up the solution of the sparse Jacobian matrix.

Requirements

- Identify the bottleneck of the current power flow analysis methods
- Hardware and software will be able to communicate with each other concurrently
- Software will run in-depth Newton-Raphson method solution
- Hardware will take inputs from software to complete Newton-Raphson method computations
- To improve runtime dramatically over pure software based computations
- Optimize radial distribution system using power solution.

Required Data

Bus Data

- Bus Number
- Bus Classification
- Voltage Magnitude
- Voltage angle
- Load Power
- Load Reactance
- Power Generated
- Reactance Generated
- Minimum Reactance
- Maximum Reactance
- Injected Shunt Reactance

Line Data

- Line Destinations (Parent Child Relation)
- Line resistance
- Line Reactance
- Line Susceptance
- Tap Setting
- Line Number
- If Line is “On” or “Off”

Calculating Jacobian Matrix

- The Jacobian is the matrix equation of the form B=Ax

$$\begin{bmatrix} \Delta P \\ \Delta Q \end{bmatrix} = \begin{bmatrix} J_1 & J_2 \\ J_3 & J_4 \end{bmatrix} \begin{bmatrix} \Delta \delta \\ \Delta |V| \end{bmatrix}$$

The diagonal and the off-diagonal elements of J_1 are

$$\frac{\partial P_i}{\partial \delta_i} = \sum_{j \neq i} |V_i||V_j||Y_{ij}| \sin(\theta_{ij} - \delta_i + \delta_j)$$

$$\frac{\partial P_i}{\partial \delta_j} = -|V_i||V_j||Y_{ij}| \sin(\theta_{ij} - \delta_i + \delta_j) \qquad j \neq i$$

The diagonal and the off-diagonal elements of J_2 are

$$\frac{\partial P_i}{\partial |V_i|} = 2|V_i||Y_{ii}| \cos\theta_{ii} + \sum_{j \neq i} |V_j||Y_{ij}| \cos(\theta_{ij} - \delta_i + \delta_j)$$

$$\frac{\partial P_i}{\partial |V_j|} = |V_i||Y_{ij}| \cos(\theta_{ij} - \delta_i + \delta_j) \qquad j \neq i$$

The diagonal and the off-diagonal elements of J_3 are

$$\frac{\partial Q_i}{\partial \delta_i} = \sum_{j \neq i} |V_i||V_j||Y_{ij}| \cos(\theta_{ij} - \delta_i + \delta_j)$$

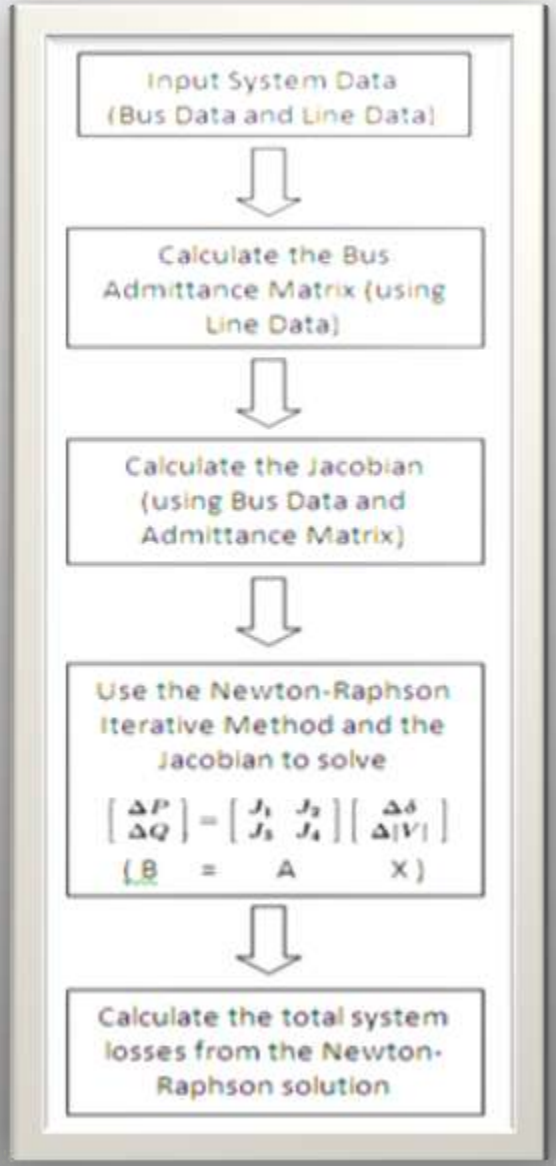
$$\frac{\partial Q_i}{\partial \delta_j} = -|V_i||V_j||Y_{ij}| \cos(\theta_{ij} - \delta_i + \delta_j) \qquad j \neq i$$

The diagonal and the off-diagonal elements of J_4 are

$$\frac{\partial Q_i}{\partial |V_i|} = -2|V_i||Y_{ii}| \sin\theta_{ii} - \sum_{j \neq i} |V_j||Y_{ij}| \sin(\theta_{ij} - \delta_i + \delta_j)$$

$$\frac{\partial Q_i}{\partial |V_j|} = -|V_i||Y_{ij}| \sin(\theta_{ij} - \delta_i + \delta_j) \qquad j \neq i$$

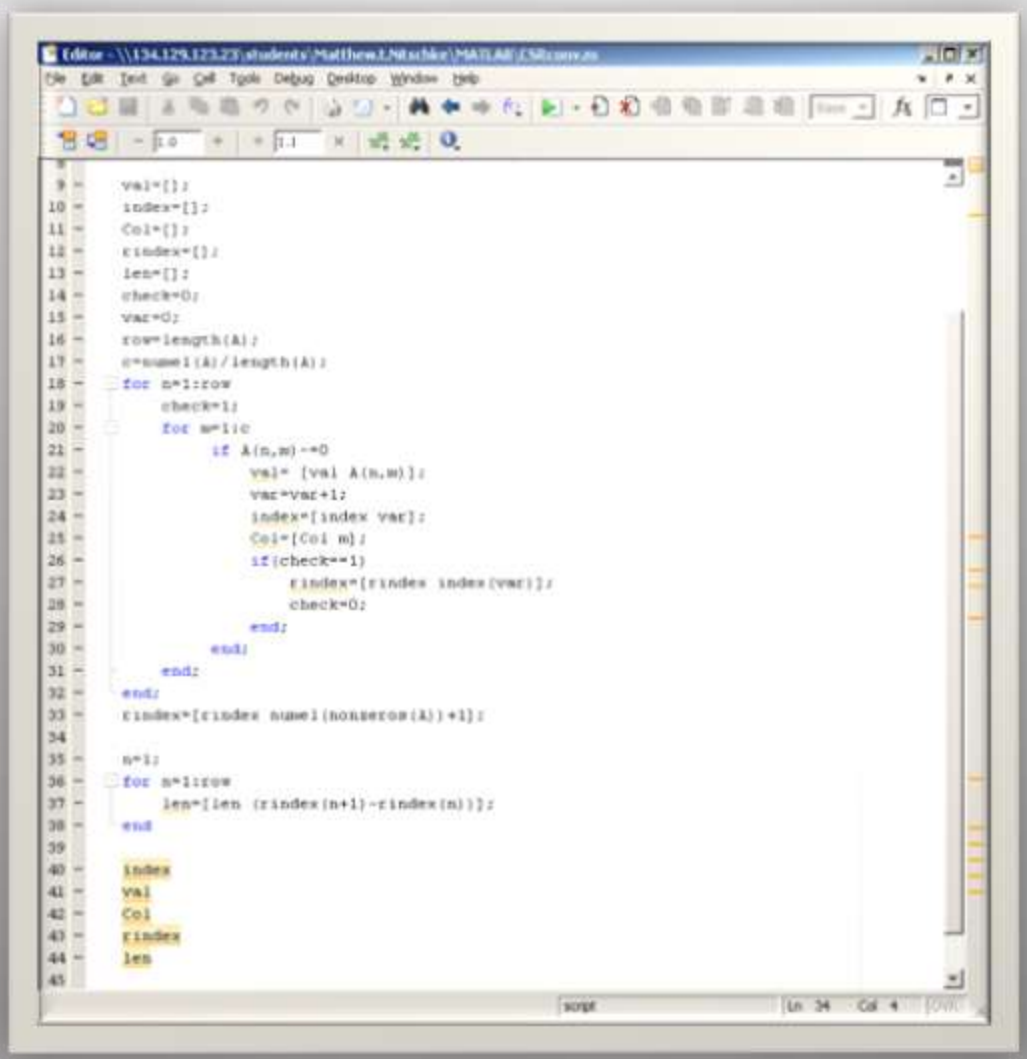
Newton Raphson Power Flow Solution Flowchart



Compressed Sparse Row Matrix Conversion

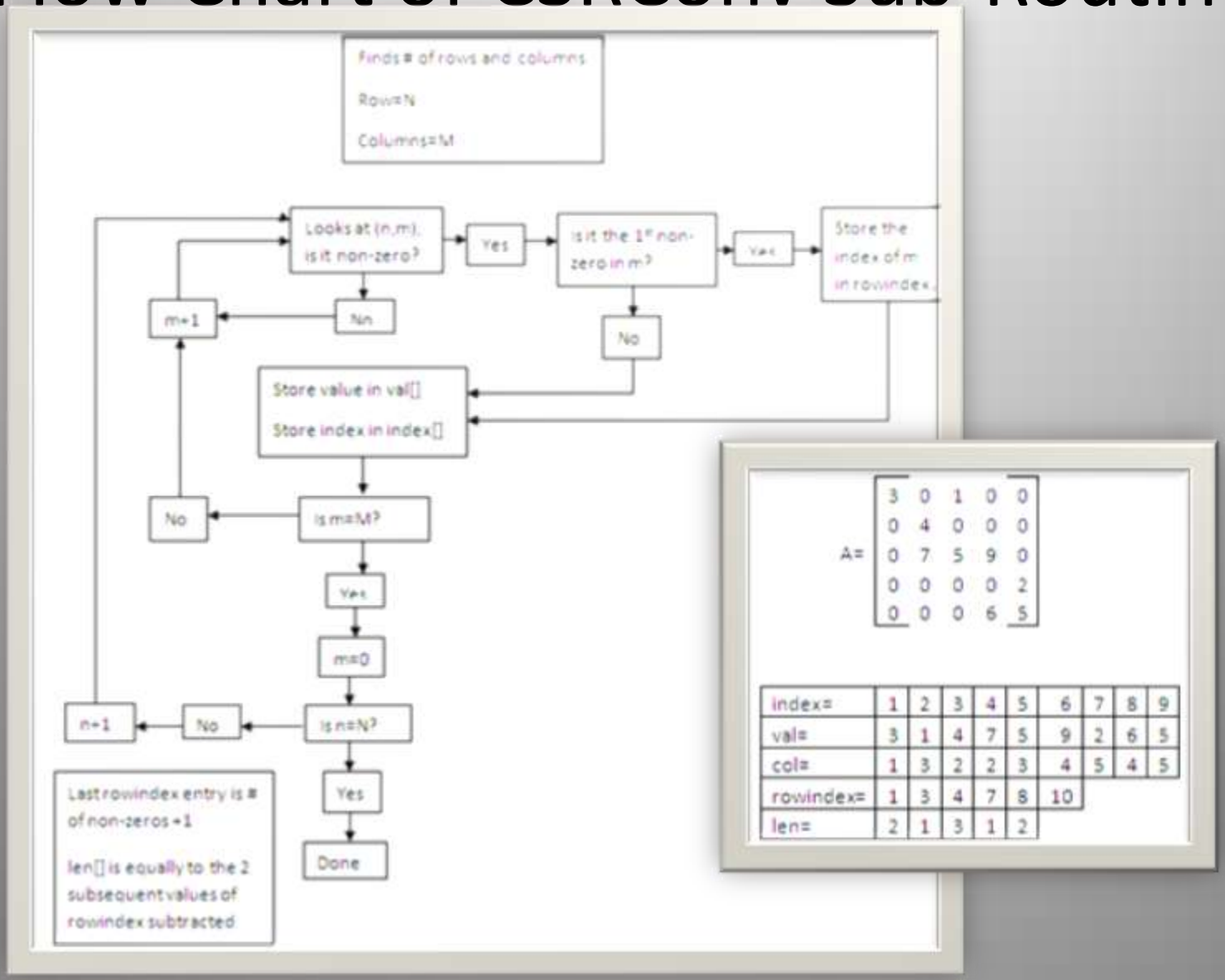
- The FPGA requires the Jacobian to be in this form in order to eliminate the zero elements of the Jacobian, that would make the Jacobian to large to handle.

CSRConv Sub-Routine



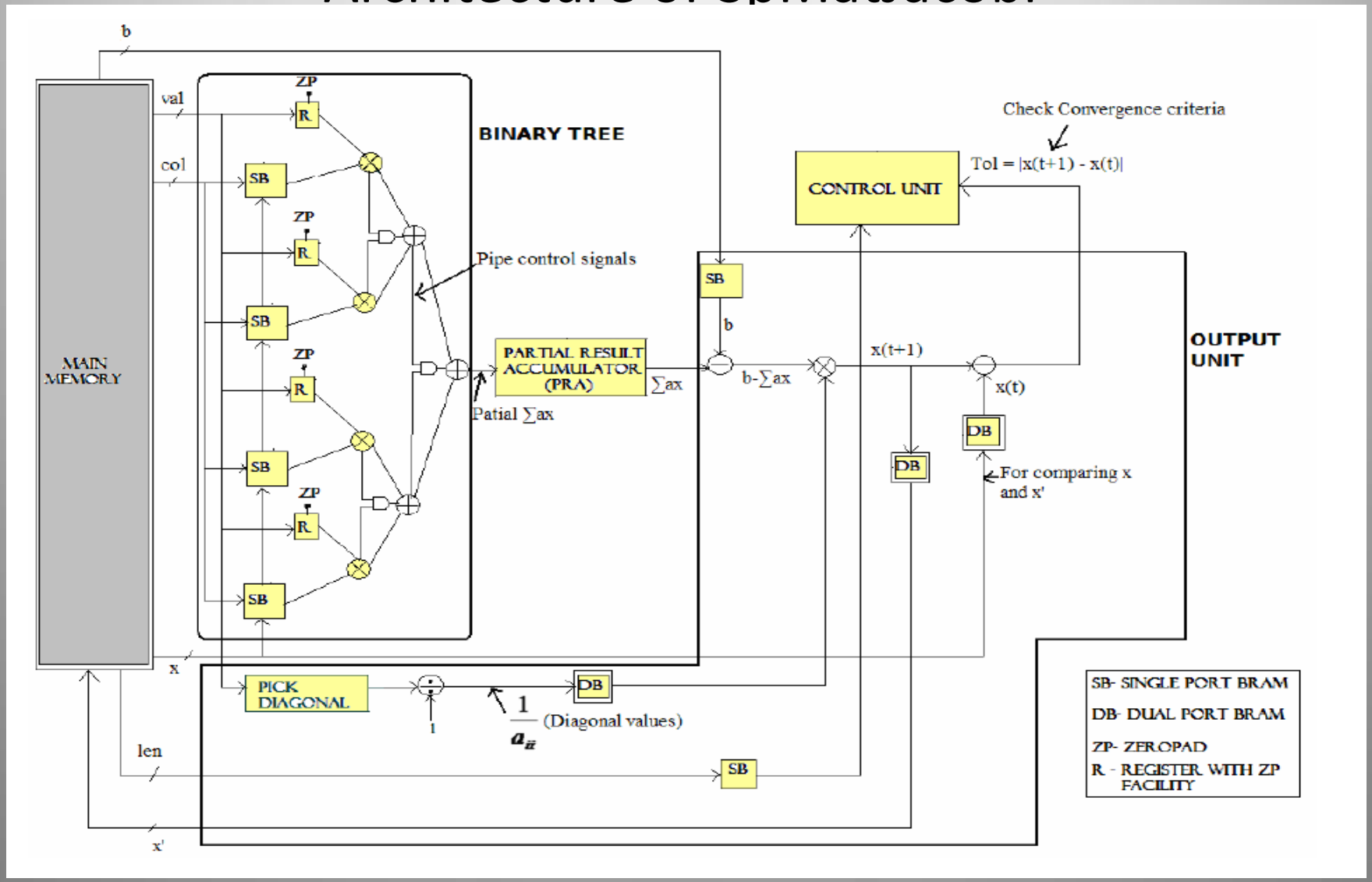
```
val=[];
index=[];
Col=[];
rindex=[];
len=[];
check=0;
var=0;
row=length(A);
c=numel(A)/length(A);
for n=1:row
    check=1;
    for m=1:c
        if A(n,m)~=0
            val= [val A(n,m)];
            var=var+1;
            index=[index var];
            Col=[Col m];
            if(check==1)
                rindex=[rindex index(var)];
                check=0;
            end;
        end;
    end;
end;
rindex=[rindex numel(nonzeros(A))+1];

n=1;
for n=1:row
    len=[len (rindex(n+1)-rindex(n))];
end

index
val
Col
rindex
len
```

Flow Chart of CSRConv Sub-Routine



Jacobi Method

Architecture of SpMatJacobi



Binary tree based Multiply Accumulate Unit

- Row is divided into multiple packets of same size and each packet is then supplied sequentially.
- Row =

2	4	5	6	10	12	-5

- Packet1 =

2	4	5	6

- Packet2 =

10	12	-5	0

- Zero is added to the last packet if not fully filled.

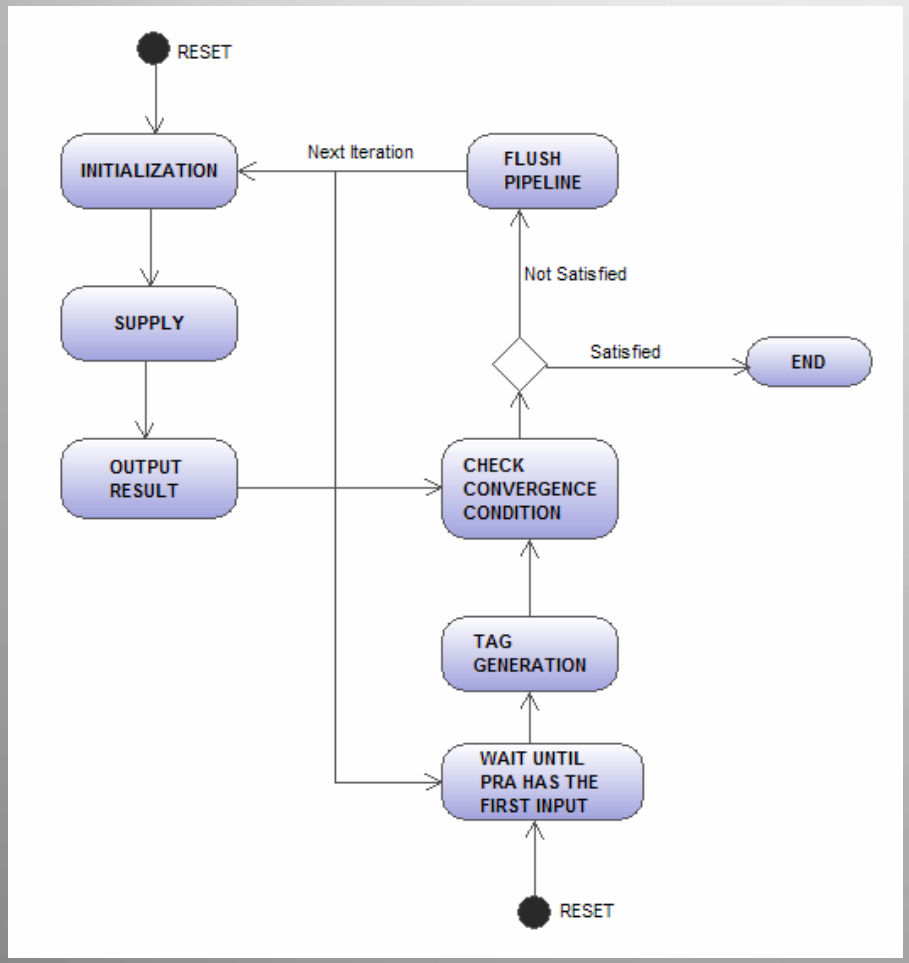
Divide Operation

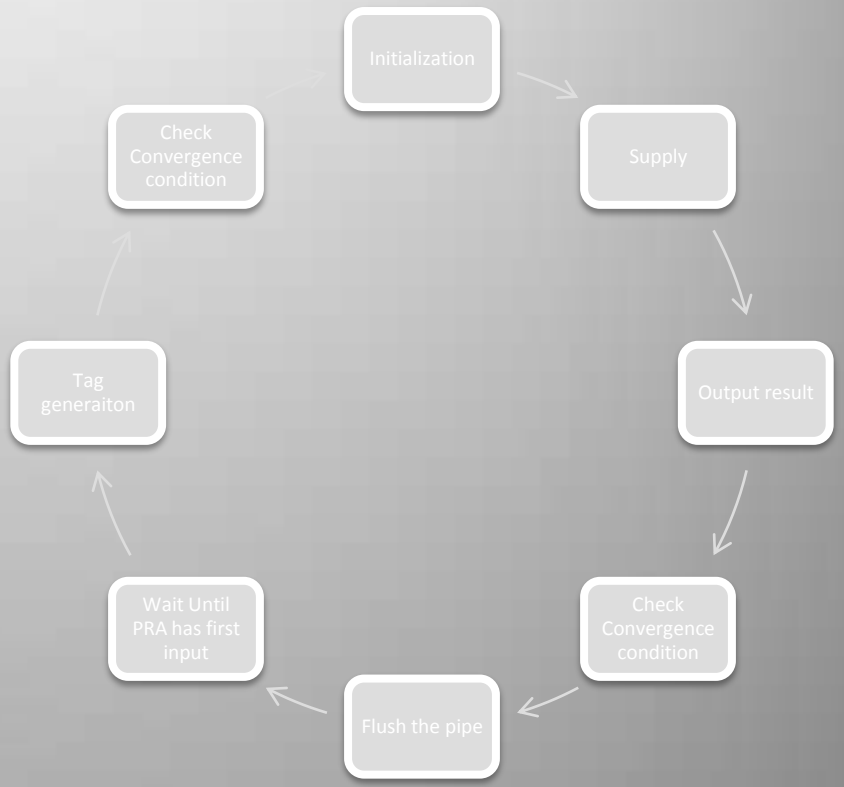
- Since Division is time consuming, it is divided into 3 steps to reduce computation time.

- 1) Calculate the reciprocal of *aii*
- 2) Calculate the numerator b - Σax
- 3) Multiply the results obtained from steps 1 and 2.

- Since Steps 1 and 2 can be performed simultaneously it reduces the overall computation time in finding x.

Control Unit





Partial Result Accumulator

Buffer(B1)

supply the partial Σax to the buffer

If(tag >= 2 level_id +1) then

If(B2 waiting for input) then

supply the value to the buffer B2

Else

supply the value to the adder

Endif

Else

If(B2 waiting for input) then

Zeropad the buffer

Endif

Supply the value to the adder

Endif

Send done to indicate that the adder can start working

Buffer(B2)

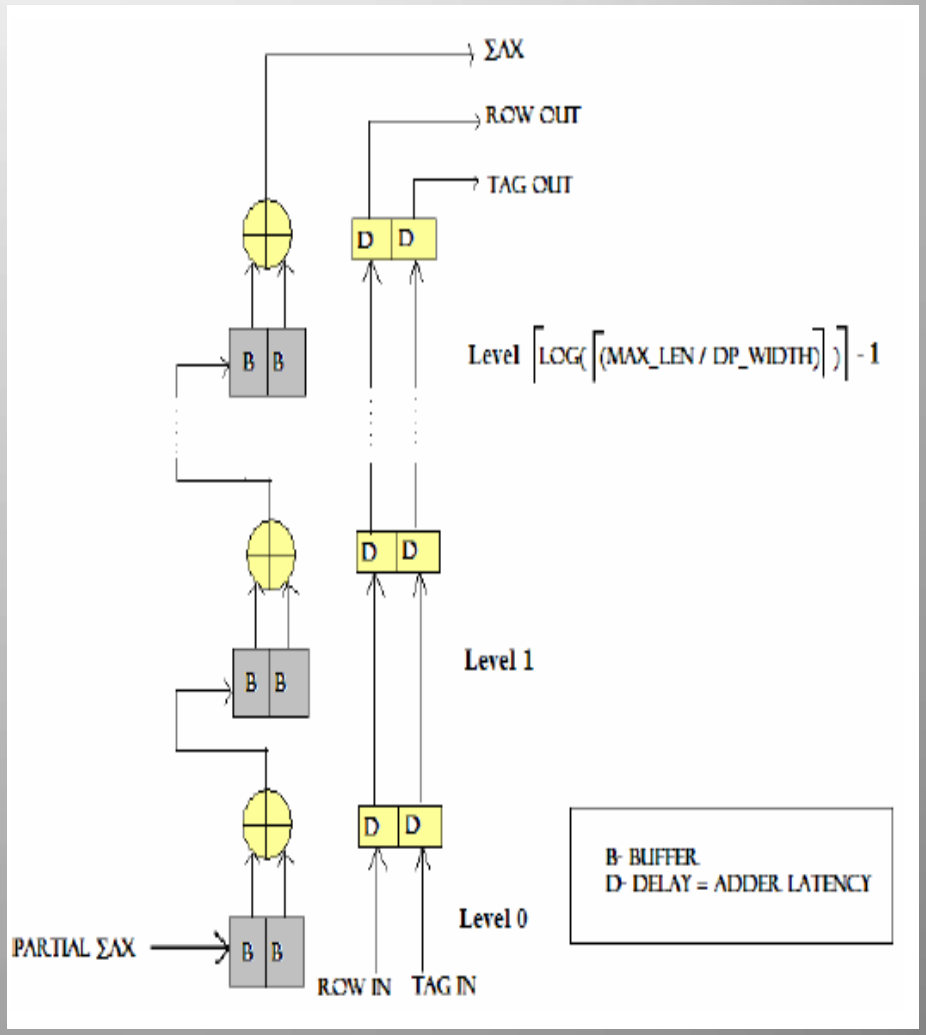
If(zeropad is false) then

Supply the value to the adder

Else

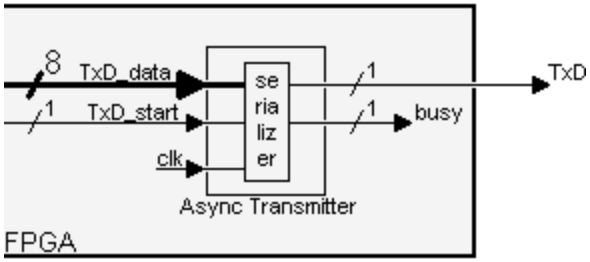
Zeropad the buffer

endif



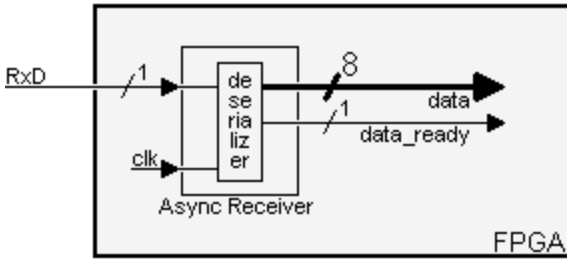
UART

Async Transmitter



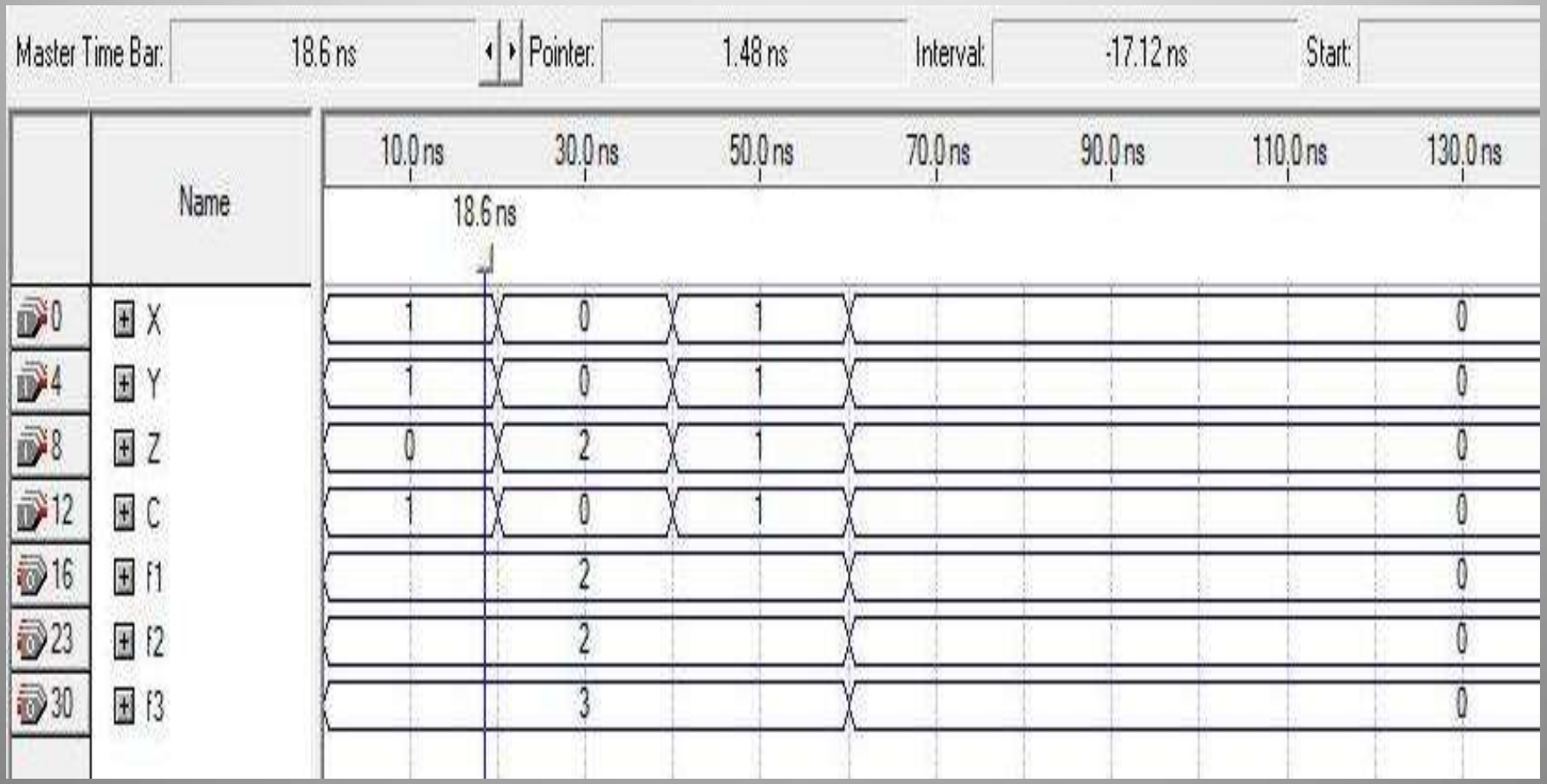

```
always @(posedge clk)
case(state)
                4'b0000: if(TxD_start) state <= 4'b0001;
                4'b0001: if(BaudTick) state <= 4'b0100;
                4'b0100: if(BaudTick) state <= 4'b1000;  // start
                4'b1000: if(BaudTick) state <= 4'b1001;  // bit 0
                4'b1001: if(BaudTick) state <= 4'b1010;  // bit 1
                4'b1010: if(BaudTick) state <= 4'b1011;  // bit 2
                4'b1011: if(BaudTick) state <= 4'b1100;  // bit 3
                4'b1100: if(BaudTick) state <= 4'b1101;  // bit 4
                4'b1101: if(BaudTick) state <= 4'b1110;  // bit 5
                4'b1110: if(BaudTick) state <= 4'b1111;  // bit 6
                4'b1111: if(BaudTick) state <= 4'b0010;  // bit 7
                4'b0010: if(BaudTick) state <= 4'b0011;  //
stop1
                4'b0011: if(BaudTick) state <= 4'b0000;  //
stop2
                default: if(BaudTick) state <= 4'b0000;
endcase
```

Async Receiver




```
always @(posedge clk)
if(Baud8Tick)
case(state)
                4'b0000: if(RxD_bit_inv) state <= 4'b1000;  //
start bit found?
                4'b1000: if(next_bit) state <= 4'b1001;  // bit 0
                4'b1001: if(next_bit) state <= 4'b1010;  // bit 1
                4'b1010: if(next_bit) state <= 4'b1011;  // bit 2
                4'b1011: if(next_bit) state <= 4'b1100;  // bit 3
                4'b1100: if(next_bit) state <= 4'b1101;  // bit 4
                4'b1101: if(next_bit) state <= 4'b1110;  // bit 5
                4'b1110: if(next_bit) state <= 4'b1111;  // bit 6
                4'b1111: if(next_bit) state <= 4'b0001;  // bit 7
                4'b0001: if(next_bit) state <= 4'b0000;  // stop bit
                default: state <= 4'b0000;
endcase
```

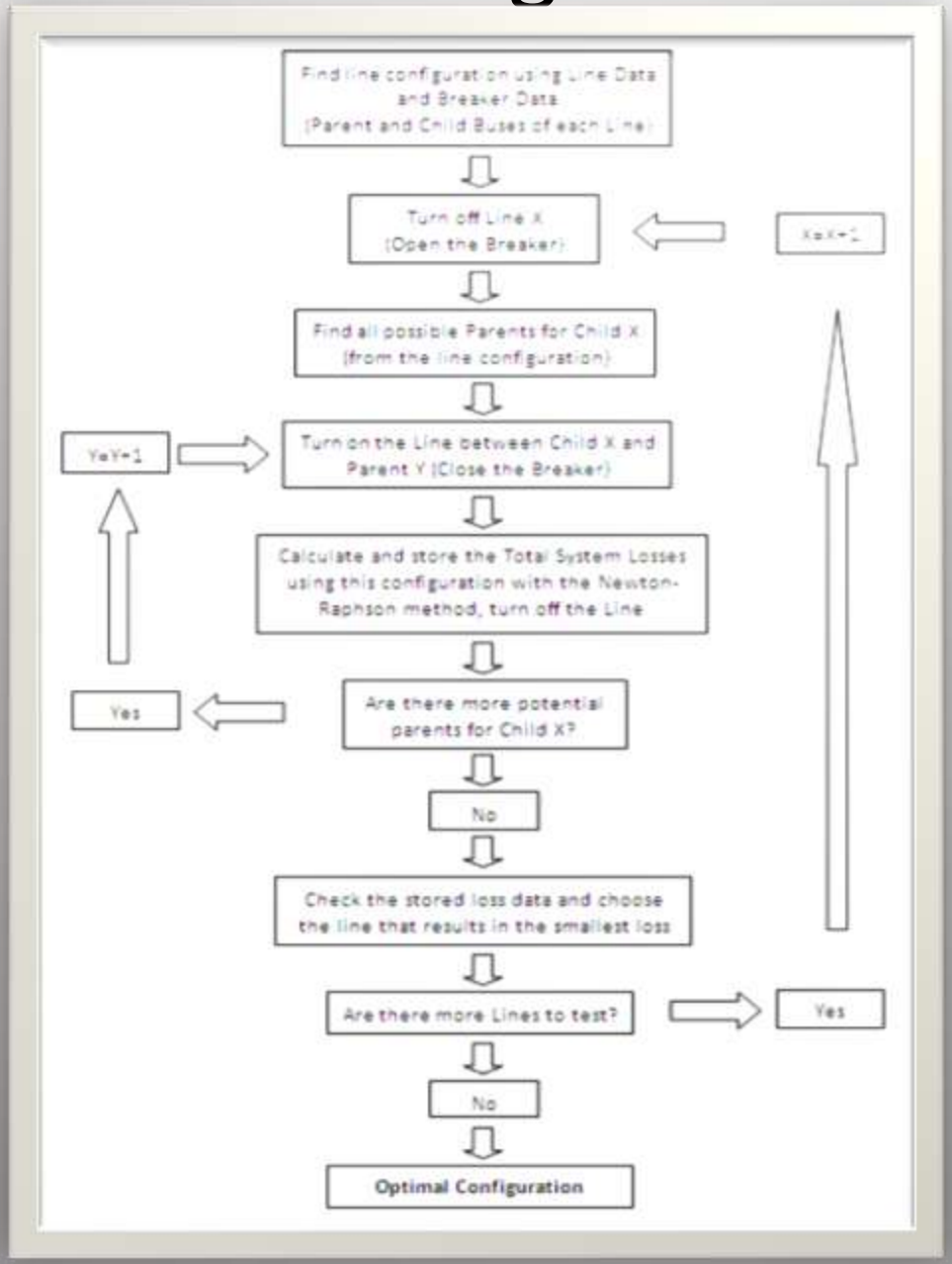
Result of 1 Iteration on FPGA



Communication Between MATLAB and FPGA

- There were several options for serial communications that we had to choose from.
  - RS-232
    - RS-232 was the favored form of communication by our advisor and is relatively simple to implement.
    - RS-232 is a common form of communication that is on most computers.
  - USB Communication
    - This form of communication is not meant for transmitting large amounts of data, due to the fact that some of the data can be lost in the communication process.
  - Ethernet
    - Ethernet is a fast form of communication but complex to implement, and was not favored by our advisor.
  - File Writing
    - We had originally planned on file writing back and forth between the FPGA and MATLAB with the TEXTIO command in VHDL. After trying to do this we found out that TEXTIO is only for simulation purposes and cannot be synthesized on the FPGA.

Network Reconfiguration Flowchart



Input Data

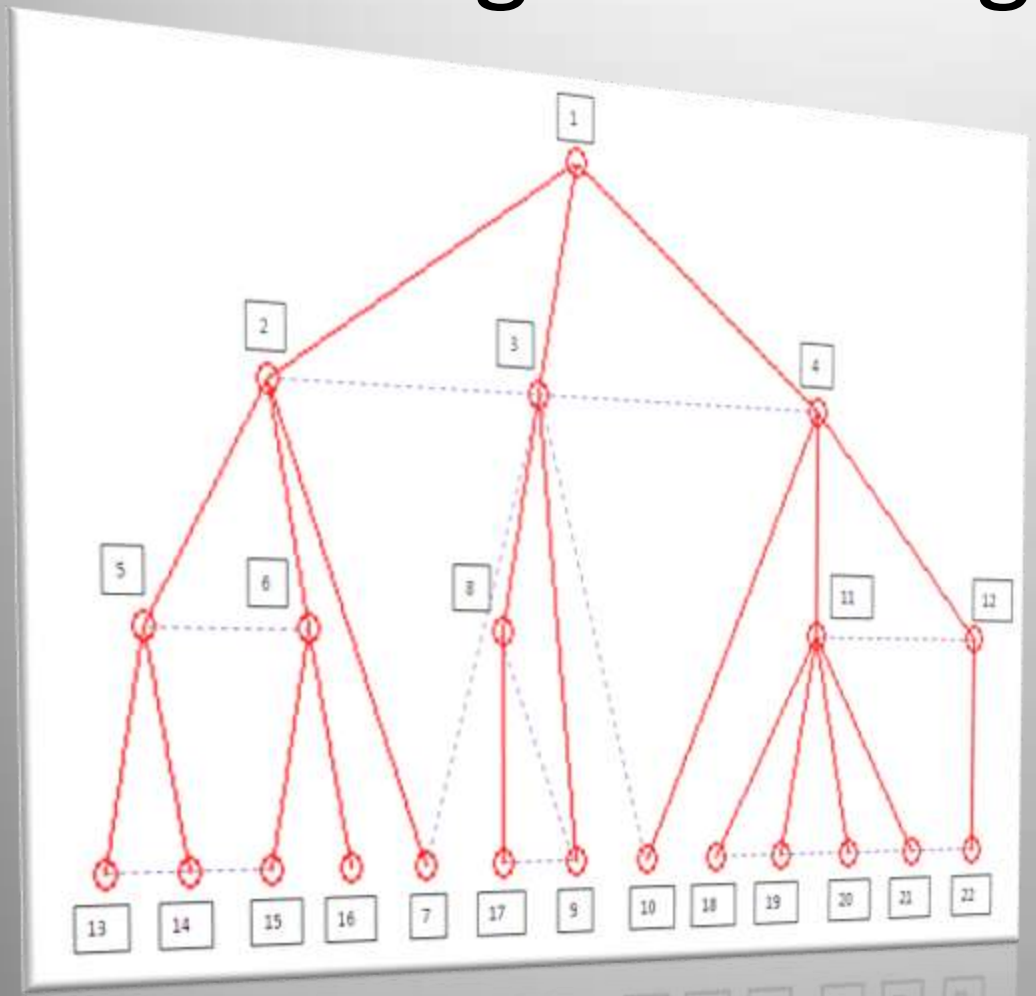
Bus Data

Bus number	Bus Classification	Voltage Mag.	Voltage Angle	Load Power	Load Reactance	Power Generated	Reactance Generated	Minimum Reactance	Maximum Reactance	Injected Shunt Reactance
1	1	1.06	0	0	0	0	0	0	0	0
2	2	1.043	0	31.7	12.7	40	0	-40	50	0
3	0	1	0	12.4	11.2	0	0	0	0	0
4	0	1.06	0	7.6	1.6	0	0	0	0	0
5	2	1.01	0	4.2	2	0	0	-40	40	0
6	0	1	0	16.2	13	0	0	0	0	10
7	0	1	0	12.8	6.9	0	0	0	0	0
8	2	1.01	0	30	30	20	0	-30	40	0
9	0	1	0	0	0	0	0	0	0	0
10	0	1	0	5.8	2	0	0	-6	24	19
11	2	1.082	0	25.6	18	0	0	0	0	0
12	2	1	0	11.2	7.5	20	0	-8	24	0
13	2	1.071	0	4.8	3.2	0	0	-6	20	0
14	0	1	0	6.2	1.6	0	0	0	0	0
15	0	1	0	8.2	2.5	0	0	0	0	0
16	0	1	0	23.5	11.8	0	0	0	0	0
17	0	1	0	9	5.8	0	0	0	0	0
18	0	1	0	3.2	0.9	0	0	0	0	0
19	0	1	0	9.5	3.4	0	0	0	0	0
20	0	1	0	12.2	10.7	0	0	-12	28	0
21	0	1	0	7.5	4.2	0	0	0	0	0
22	2	1	0	16.8	12.4	0	0	-12	40	0

Line Data

Line Destination		Resistance	Reactance	Susceptance	Tap Setting	Line Number	Breaker
1	2	0.0192	0.0575	0.0164	1	1	1
1	3	0.0452	0.1852	0.0204	1	2	1
1	4	0.057	0.1737	0.0184	1	3	1
2	3	0.0132	0.0379	0.0042	1	4	0
2	5	0.0472	0.1983	0.0109	1	5	1
2	6	0.0581	0.1763	0.00476	1	6	1
2	7	0.0119	0.0414	0.02001	1	7	1
3	8	0.046	0.116	0.0102	1	8	1
3	9	0.0267	0.082	0.0085	1	9	1
3	4	0.016	0.042	0.0045	1	10	0
3	7	0	0.208	0	0.978	11	0
3	10	0	0.556	0	0.969	12	0
4	10	0	0.208	0	1	13	1
4	11	0	0.11	0	1	14	1
4	12	0	0.256	0	0.932	15	1
5	6	0	0.14	0	1	16	0
8	9	0.1231	0.2559	0	1	17	0
11	12	0.0662	0.1304	0	1	18	0
5	13	0.0945	0.1987	0	1	19	1
5	14	0.221	0.1997	0	1	20	1
6	15	0.0824	0.1923	0	1	21	1
6	16	0.1073	0.2185	0	1	22	1
8	17	0.0639	0.1292	0	1	23	1
9	17	0.0936	0.209	0.00395	1	24	0
11	18	0.0324	0.0845	0	1	25	1
11	19	0.0348	0.0749	0	1	26	1
11	20	0.0727	0.1499	0	1	27	1
11	21	0.2116	0.0236	0	1	28	1
12	22	0.1	0.202	0.00402	1	29	1
13	14	0.115	0.179	0	1	30	0
14	15	0.132	0.27	0.00119	1	31	0
18	19	0.1885	0.3292	0.00576	1	32	0
19	20	0.1093	0.2087	0	1	33	0
20	21	0.063	0.396	0	0.968	34	0
21	22	0.3202	0.6027	0.0102	1	35	0

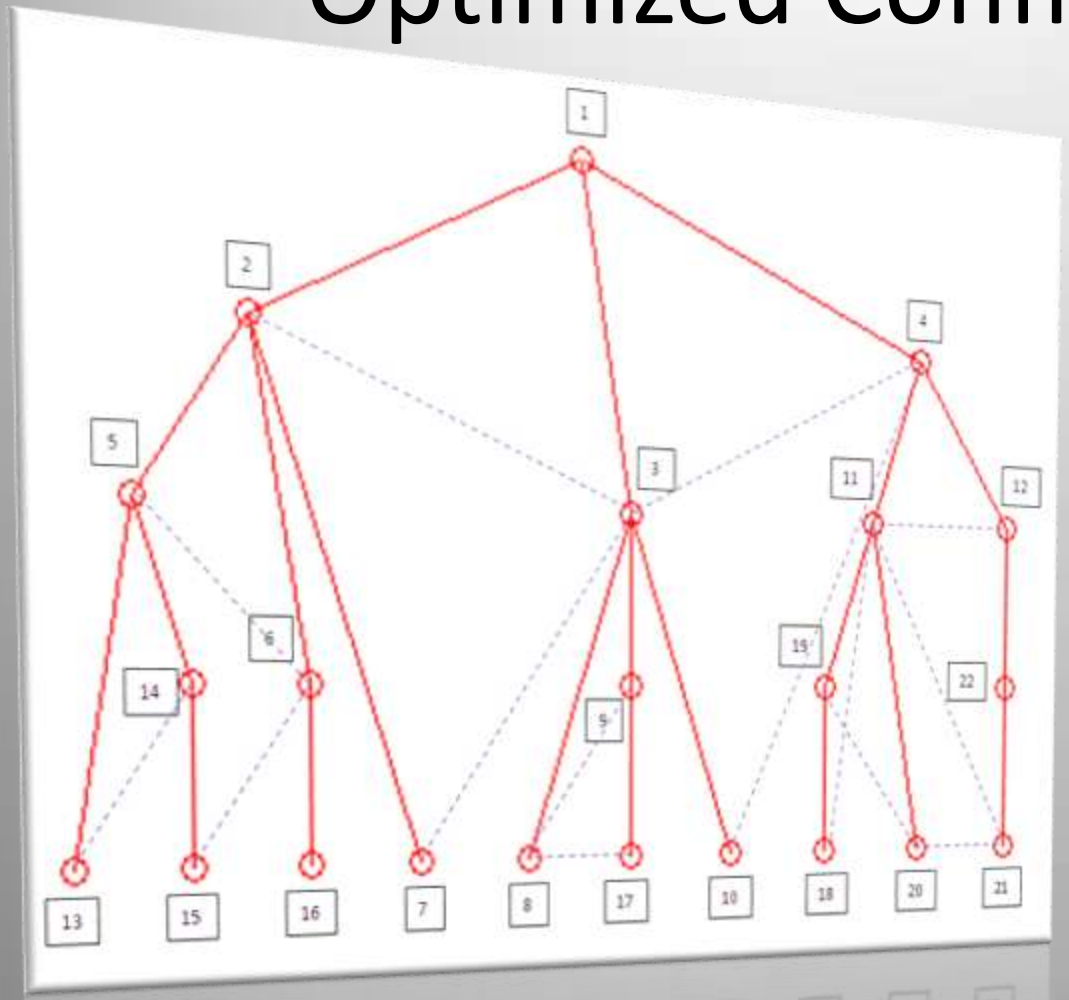
Original Configuration



Original losses:
9.889 MW
11.835Mvar

Total MVA Loss:
15.4229MVA

Optimized Configuration



Optimized losses:
10.150 MW
7.872 Mvar

Total MVA Loss:
12.8455 MVA

Problems Encountered and Lessons Learned

- UART not sending correct data
  - Our UART was sending incorrect values (i.e. 3 sent and 3.624 received or ASCII characters received)

- RS-232 is a relatively slow form of communication
  - It takes several seconds to send one data set through RS-232.
  - We needed the communication to be much faster to speed up the power flow solution

- Asdf

Final Budget

Part	Quantity	Retail Cost	Expected Cost	Total Cost	Notes	Spent to date
Altera FPGA Development Board	2	$650	$0	$0	1 Board provided by Advisor	$0
Hadi Sadat Power System Analysis book	1	$160	$160	$160		$0
MATLAB License	2	$100	$100	$200		$0
Misc.	1		$72	$72		$0
	Total Retail	$910	Total Expected	$432	Total to Date	$0

All of our components were provided to us, so we did not have to spend any of our allotted budget.

Future Work

- Mount FPGA on the same board as the processor, to allow for direct communication.
  - Without RS-232 the communication time can be cut down drastically.
- Take real world inputs (i.e. CT and VT inputs) and substitute them in the bus data and breaker data.
- Make a test board to test the optimization option of the project.

Summary

- Overall goal: Decrease the time it takes to run the Newton-Raphson power flow solution for pure software and optimize a radial distribution system.
- Problems with the speed of RS-232 communication makes the initial goal of hardware acceleration unachievable given the constraints of the project.
- Future Work
  - Mounting FPGA on processor board
  - Using real world inputs
  - Testing in real world situations